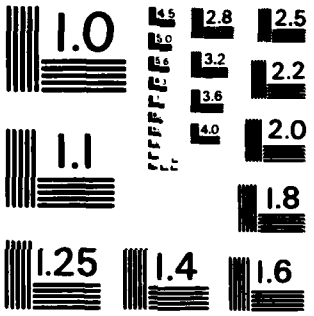
AD-A145 408 SUPERCOMPUTERS AND VLSI: THE EFFECT OF LARGE SCALE 1/1
INTEGRATION ON COMPUTE..(U) WASHINGTON UNIV SEATTLE
DEPT OF COMPUTER SCIENCE L SNYDER AUG 84 TR-84-08-05
UNCLASSIFIED N00014-84-K-0143 F/G 9/2 NL

END
DATE FILMED
10 84
DTIC



(12)

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-84-08-05	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Supercomputers and VLSI: The Effect of Large Scale Integration on Computer Architecture		5. TYPE OF REPORT & PERIOD COVERED Technical, interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Lawrence Snyder		8. CONTRACT OR GRANT NUMBER(s) N00014-84-K-0143
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Washington Department of Computer Science FR-35 Seattle, WA 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Task SRO-100
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Information Systems Program Arlington, VA 22217		12. REPORT DATE August 1984 13. NUMBER OF PAGES 17
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Parallel computation, parallel architecture, planarity, VLSI architecture, CHiP computer, single chip computer.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The use of VLSI technology to build supercomputers is analyzed in depth. The benefits of VLSI are reviewed, and the liabilities are explored thoroughly. The perimeter problem and the planarity problem are identified as being critical limits on architectural design. The CHiP architecture, a highly parallel computer designed with VLSI implementation in mind, is scrutinized in terms of how well it exploits the benefit of VLSI and how well it avoids the liabilities. Not surprisingly, it does pretty well.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Supercomputers and VLSI: The Effect of Large-Scale Integration on Computer Architecture

LAWRENCE SNYDER[1]

Department of Computer Sciences
Purdue University
West Lafayette, Indiana

The mere possibilities inherent in very large scale integration (VLSI) do not themselves create adequate ways of exploiting this technology.

J. T. SCHWARTZ (1980)

[1] Present address: Department of Computer Science, FR-35, University of Washington, Seattle, Washington 98195.

ADVANCES IN COMPUTERS, VOL. 23

ISBN 0-12-012123-9

1. Introduction

A supercomputer is the largest, fastest computer available at any given time. It is the antithesis of the single-chip microprocessor. But the microprocessor has captured the spotlight of very-large-scale integration, even though the technology's benefits will be most clearly seen by its impact on supercomputers. The benefit is not in the prospect of a one-chip supercomputer; indeed, it will be argued that a one-chip computer, super or otherwise, is a doubtful result of VLSI technology. Instead, the benefit is in the prospect of imaginative, new architectures, in being relieved of the tight constraints that have bound computer design for decades.

In order to find creative ways of exploiting the technology, we must understand the benefits it offers and appreciate the limitations it imposes. The assets and liabilities of VLSI will be enumerated in the subsequent discussion to prepare the reader for an assessment of one VLSI architecture, the Configurable, Highly Parallel (CHiP) computer. Along the way principles will be enunciated and useful forms of analysis will be demonstrated.

To begin, we set the course with an excursion into familiar territory: the microprocessor. There are two objectives. First, without presuming a knowledge of VLSI, we wish to provide a realistic idea of how much circuitry fits on a chip. (For an excellent introduction to VLSI, see Clark, 1980.) This will be done in functional or architectural terms. Second, we wish to accustom ourselves to thinking about computers as VLSI chip designers and computer architects do. This will prepare us for scrutinizing VLSI implementations of various architectures in subsequent sections.

2. The Single-Chip Computer

The dream of a single-chip computer has been discussed ever since significant circuit integration became a reality more than a decade ago. The dream has not yet been realized, and depending on the particular definition of "computer," it may not be.

To be sure, single-chip microprocessors have been on the scene for a few years, first in an 8-bit form, and then in rapid succession in 16- and 32-bit versions. These are not "computers" in the sense of being the totality of the electronics of a computer system such as a personal computer; they are the central processing units of such systems. There is little or no on-chip random-access memory (RAM). To qualify as a computer in this sense, the microprocessor chip must be coupled with a substantial random-access memory, perhaps a floating-point processor, and input/output control facilities. The microprocessors of today are just that, processors, and a variety of other chips must be added to build a general-purpose computer.

It would seem that to make a one-chip computer requires only an improvement in VLSI technology sufficient to allow all the chips of a current machine to fit on one piece of silicon. Such technological improvements are taking place, but the result will probably not be a single-chip general-purpose computer. To see why, consider some history.

The first microprocessors were 8-bit machines, that is, the unit of data that the processor could use, the data path width, was 8 bits. There have been substantial technological improvements in VLSI since those early machines, making more transistors available to the designer. This added circuitry could have been used for on-chip RAM, or it could have been used for logic to improve the functionality or performance of the microprocessor.

Used as on-chip RAM the technological improvements would have yielded a tiny, self-contained computer. This machine would be an endpoint of the computer continuum that runs from the simplest stored program device to a supercomputer. To some extent the self-contained computer has been implemented for cases where it is important to have a single package, for example, when the microcomputer is to be embedded in another device for a fixed control or monitoring application. In such cases the RAM is almost insignificant, and the fixed program is stored in read-only memory (ROM). Actually, these devices tend to be an easy way to implement functions that would otherwise require complex electronic design, and they do not qualify as computers in the sense used here.

Alternatively, the added circuitry could be used to give a more powerful processor chip which, when coupled with other chips, could make a more powerful general-purpose computer. Because chips are inexpensive and a single package is typically unimportant for general-purpose computing, the designers favored the option of increased capability. This greater capability took the form of the wider data path of the 16- and 32-bit microprocessors.

To see that the trend toward microprocessors with greater capability was not a historical quirk but rather was based on architectural realities, we hypothesize further technological improvements in VLSI, and speculate where they might lead.

Postulate a 32-bit microprocessor with no on-chip RAM, i.e., today's technology. Further, suppose that VLSI fabrication technology permits a reduction in the feature size by a factor of two. Feature size is the width of the smallest element on a chip, e.g., a wire, and is used to scale all elements on a chip (Mead and Conway, 1980). Because VLSI chips are two-dimensional, the scaling applies in both directions, and the device density, the number of transistors that fit on a chip, increases by a factor

of four. Thus only one-quarter of the area of the new chip is devoted to the processor. That leaves three-quarters of the chip for additional circuitry. How will it be used? Again we choose between added logic or on-chip RAM.

Additional logic can be used to make the processor more functional or to improve performance. Functionality can be increased by enhancing the instruction set—the typical microprocessor has a fairly rudimentary instruction set—or by adding features such as floating point or other data types. For example, a low-performance floating-point processing unit would require roughly the same area as the postulated 32-bit microprocessor, thus with the fourfold density improvement it should also require about a quarter of the new chip. Logic, used to improve performance, might take the form of virtual-memory address-mapping hardware, instruction-stream pipelining, etc. In any case, the result would be a more powerful processor. When this is coupled with the new, denser RAM chips, which would also be made possible by the factor-of-four improvement in technology, the result would be a more powerful computer system.

Alternatively, staying with the old microprocessor and adding on-chip RAM will give a weak processor with a memory only *three-fourths the size of a single memory chip.* This will move us out farther on the computer continuum than the previous one-chip machine, but it will be a long way from the multichip computer. In particular, when considering the memory needed for software to support general-purpose computing, this would be a very meager computer indeed.

Thus the trend toward more capable microprocessor chips was probably not a quirk, but rather a consequence of the fact that chips are small and inexpensive, and the benefits from additional chips generally outweigh their costs. It seems likely, then, that for the next few years, it will be the processor alone, not the whole computer, that occupies the processor chip. Whether this trend continues through subsequent improvements in VLSI depends on many factors. It is too risky to speculate further. Suffice it to say that there are a variety of other considerations, architectural, technological, and pragmatic, beyond the simple logic/RAM tradeoff just discussed that militate against a one-chip computer. These will be considered in the next three sections as we consider the advantages, disadvantages, and architectural considerations of VLSI.

3. The Advantages of VLSI

The benefits of VLSI technology will strongly influence the organization of computers, as has already been suggested. In this section we identify those benefits in preparation for our subsequent analysis of how VLSI might be used to build innovative architectures.

3.1 Economics

At the heart of the one-chip computer argument is cost. Chips are cheap. It is sensible to continue to choose multichip systems as long as the technological enhancements raise device density without raising the cost of the chips by a corresponding amount.

It is important to emphasize the source of this high-value, low-cost benefit. It derives from the fact that the circuit is transferred to the semiconductor material by means of a lithographic process (Clark, 1980; Mead and Conway, 1980). (Traditionally, photolithography has been used, but, with the desired feature size approaching the wavelength of light, other methods, e.g., X-ray lithography, are being developed.) As light passes through the mask (i.e., the "negative" that contains a photograph of the circuit) to expose a photosensitive "dope" on the substrate, the entire circuit design is transferred to the chip at once. The radiational or broadcast character of light transfers the design independent of its complexity. Thus, just as the cost of printing a page is independent of the number of words it contains, the labor to fabricate a chip is essentially independent of the number of transistors used. This permits manufacturing techniques that maintain the low cost.

The chips are not fabricated individually, but are clustered together into a single, circular wafer of 3–5" in diameter. This permits more than 100 chips to be handled as a unit. The process of converting a silicon disk into a wafer of chips is long and exacting. There are seven mask layers in a typical nMOS[2] process and each involves steps such as depositing a material onto the wafer, "doping" it, transferring the mask design, and etching away the exposed regions to remove the unwanted material. Each mask step must align perfectly so that the deposited materials form a composite "picture" that is the circuit. In addition to alignment errors, the process is fraught with other hazards—incomplete depositions and etches, specks of dust, scratches, etc.—so not every chip on the wafer will be perfect. Therefore, the yield, the number of good chips as a percentage of the total produced, can be very low, especially while a new process is coming online. Consequently, the price of the highest technology chips will initially be high. However, as the bugs are shaken out of the "fab line," the yield will rise, and this together with competition tends to reduce the price.

[2] nMOS is an abbreviation for "n-channel metal–oxide–semiconductor"; n, in turn, is an abbreviation for "negative"; for a positively engaging explanation of why anyone would want to abbreviate these terms, see Clark (1980).

3.2 High Integration

Another benefit of VLSI which sounds almost tautologous is the high degree of integration. The semiconductor material can be used for active logic components, memory, and wires in any combination or arrangement. This property is exploited in the microprocessor chip design where there is logic for the arithmetic, memory for registers, and wire woven throughout. Interestingly, however, high integration is used much less extensively in the overall computer design. If we consider all the chips making up a sequential computer, one processor chip, some auxiliary chips, and many memory chips, we find a marked imbalance in the distribution of logic. To be sure, there is some logic on a memory chip, but most of the system's logic is concentrated on the processor chip and the I/O controlling chips.

Even though this logic imbalance is simply an artifact of the von Neumann architecture (see Section 5), balance is an important point to keep in mind when considering other architectural designs. Processing or data transformation will take place in the logic-rich areas of the machine. The data will be concentrated in the logic-poor memory. To adopt a diffusion metaphor, computation requires the movement of data from areas of higher concentrations to areas of lower concentration (logic). If there are few such logic-rich sites, the diffusion (processing) rate must be low. We might raise the rate by more widely distributing the logic. This is another way of motivating parallelism and emphasizing that the technology favors parallelism.

3.3 Density

The higher density of VLSI favors reliability and therefore makes it realistic to have a system with millions of gates (Lincoln, 1983). Reliability is improved not only because fewer chips are needed for a given number of gates, but also because the cost of a gate is lower. Thus, devoting some gates to error correction and testing circuitry does not engender a great cost in terms of the loss of usable gates.

A word about potential density improvements is in order at this point. We have assumed throughout the discussion that in the future it will be possible to place still more transistors on a chip. This is almost a certainty. However, the continuation of the Moore curve (Moore, 1979)—the trend for device density to double every 2 years (see Fig. 1)—is the subject of considerable debate. Obviously, this exponential increase cannot continue indefinitely. (An increase of any slope cannot be sustained indefinitely, of course, because there are physical limitations on the size of circuits. What is at issue is the continuance of the exponential portion

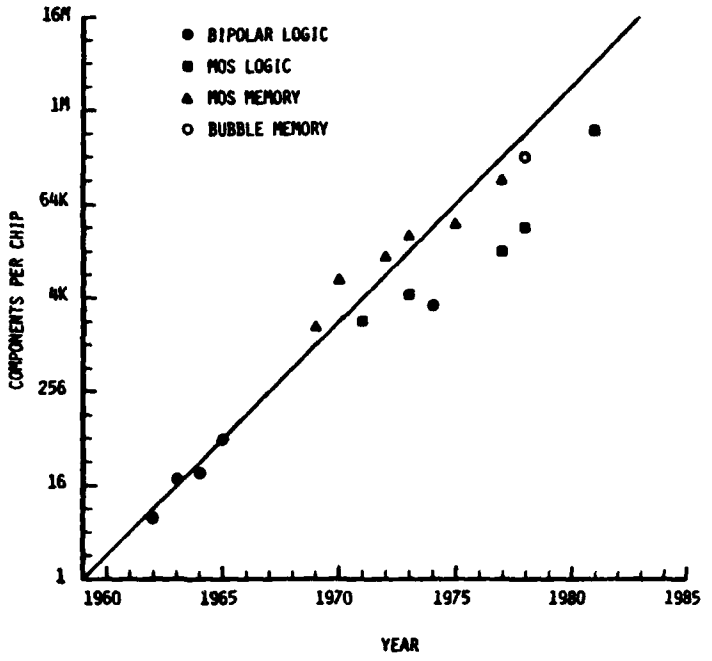


FIG. 1. The "Moore curve."

of the curve.) Some researchers believe the curve is already flattening out; others disagree, but acknowledge that the trend will end soon. The prediction is difficult because a number of factors come into play.

Traditionally, device density is improved by reducing the feature size, reducing the size of the transistors and the width of the wires. We will call this "technological density improvement." With each feature-size reduction, the fabrication process becomes considerably more exacting. The clean room must be cleaner; the physical limitations of the materials and processes involved become more constraining (Swensson, 1983). It is harder to keep the yield economically high.

But there is another technological means of increasing the number of transistors on a chip. The chip can be made larger. This increases the chance that a speck of dust or a processing anomaly will render the chip dysfunctional, because the area of the chip is larger. However, fault tolerance can be designed into the chip, as is now being done with high-density memory chips. In the limit, the size of the chip gets so large that it ceases to be a chip. Wafer-scale integration is a topic of active research that

seeks to provide dramatic technological density improvements (Hedlund and Snyder, 1982; Raffel *et al.*, 1980).

Another means of raising the effective device density is available to the computer architect: design simplification. We call this "architectural density improvement." A chip designed to do a particular function and using a given technology can *appear* to exhibit an increased device density if the design is simplified and if the freed area is used to provide additional functionality or improved performance. Design simplification can be achieved simply by ingenuity. There is also a methodological approach to design simplification: *eliminate the little-used or redundant components of a design*. This is especially useful to the architect because it is based on a global, or system-wide, examination of a design.

The Reduced Instruction Set Computer (RISC) is an example of how this principle has been applied to microprocessor design (Fitzpatrick *et al.*, 1981; Patterson and Séquin, 1981). By analyzing the frequency of various instructions output by a compiler, the RISC designers identified a small set of critical instructions. Their processor supports only those (31) instructions. The result was a substantial savings in control logic, down from 50% of the chip area on a typical microprocessor to 6% on RISC I. The saved area could then be used for added internal registers (78 on one design, 138 on another) and a full 32-bit data path. Even though the instruction set was limited, the RISC approach probably represents a net increase in functionality, as indicated in Table I (Foderaro *et al.*, 1982).

A single instruction stream, multiple data stream (SIMD) architecture such as ILLIAC IV (Bouknight *et al.*, 1972) is an example of redundancy elimination in the parallel-processing context. All processors execute the same instructions broadcast to them from a single source. The obvious

TABLE I

COMPARISON OF RISC I WITH THREE MICROPROCESSORS ON FOUR PROGRAMS

		Program time (milliseconds)			
Machine	Language	Search	Sieve	Puzzle	Acker
8086	Pascal	7.3	764	44,000	11,100
iAPX-432	Ada	4.4	978	45,700	47,800
MC 68000	C	4.7	740	37,100	7800
Average		5.5	827	42,300	22,200
RISC I	C	2.5	698	23,500	16,000

savings is in program memory and instruction fetch and decoding logic because there is only one copy of these components for the whole machine.

There is a secondary simplification possible in designing the instruction interpretation logic for SIMD machines. If the instructions are "high-level," expressive instructions, each processor requires considerable logic to interpret the instruction. If it has "low-level," rudimentary instructions, closer to microcode, then there is less interpretation logic. This approach has been used in the design of the 16K processor computer called the Massively Parallel Processor (Batcher, 1980).

There are other benefits of VLSI having to do with power requirements, physical size, etc. Most of these have little influence on the architectural decisions. The key benefits are low cost, a high degree of integration, and the potential for further increases in the total number of gates.

4. The Constraints Imposed by VLSI

If the potential benefits of VLSI are enormous, the constraints it imposes are equally enormous. In this section these constraints are catalogued and their consequences explored. When the analysis is over, the benefits will likely be judged to outweigh the constraints, but the reader should decide.

4.1 Regularity

A supercomputer might contain as many as 100,000 components, a significant proportion of which will be VLSI chips. As previously noted, mass-production manufacturing techniques will permit the chips to be fabricated relatively inexpensively, but the fabrication cost is the variable cost; the fixed cost is the design, or mask development cost. Masks are designed by artisans and they are expensive to produce, especially in terms of time (Moore, 1979). Clearly, this design cost is amortized over the units produced, so the time to develop and the expense to produce a system's chips grow with the number of different types of chips used. This imposes a constraint of *regularity* on the architecture; some chips must be used extensively. Moreover, it is desirable that the regularity extend to the "board level" because it is subject, though to a lesser degree, to the same considerations.

The design cost for the chip masks imposes the regularity constraint. There are both technological and architectural ways to reduce design cost.

The main technological strategies for reducing the number of distinct masks required for a system are based on a common principle: *develop a general design that can be specialized to a variety of uses*. Typically, the specialization process is partially or entirely automated to lessen the effort of the designer. Examples for custom VLSI design include program logic arrays (PLAs) (Mead and Conway, 1980), general circuits that can be automatically generated from logic equations, and "silicon compilers" (Siskind *et al.*, 1982) that generate mask layouts from "high-level" specifications. Another variant is to develop a processor whose control logic is given by a ROM; the mask for a particular process can be automatically generated by a program that codes the ROM. The gate array (Werner, 1981) is still another alternative. It is a chip designed with cells containing commonly used logic functions, for example, NANDs and NORs, and with predetermined channels for routing wires. These chips are fabricated without the logic functions being connected together. Later, a designer specifies a wiring scheme that connects the components to achieve the desired function. Then, in another fabrication step, metal wires are added to implement the wiring scheme. In every case the design cost is reduced through automation.

The architectural means for meeting the regularity constraint would likely take the form of a computer built of repeated substructures. Such machines are sometimes called homogeneous and there are several examples either built or proposed. The Massively Parallel Processor (Batcher, 1980) is an architecture with 16K processors designed for image processing. The small processors and their registers have been packed eight to a chip (see Fig. 2), thus the design cost is amortized over 2K units. Systolic arrays (Kung, 1982) are homogeneous special-purpose computers composed of an array of small processors (see Section 5.2.1). They satisfy the regularity constraint very well—they are built from essentially one type of chip. (Depending on the intended algorithm, systolic arrays sometimes require a few chip types, but these are generally quite similar and the variety could be handled by technological means.) Still another regular architecture is the Configurable, Highly Parallel (CHiP) architecture (Snyder, 1982), to be discussed in Section 6. Note that these examples of meeting the regularity constraint reflect a high degree of system integration.

4.2 Planarity

Perhaps the most significant constraint imposed by VLSI is that it is a two-dimensional technology. The constraint is further magnified by the fact that the chips are usually placed on two-dimensional boards. Exacerbating the problem a bit more, the boards are usually placed in a cage or rack and connected on one side by a "back plane." The two-dimensional constraint is not a *planarity* constraint in the sense that wires are not permitted to cross over each other. VLSI technologies typically permit one or two levels of crossover and boards permit several levels of crossover. The back plane permits a significant degree of crossover as well as the possibility of violating two-dimensionality altogether by connecting the boards at other places; it would be inappropriate, however, especially for a large system, to consider this to be truly three-dimensional. The two-dimensional constraint is a significant consideration, and for brevity we refer to it as the *planarity problem*.

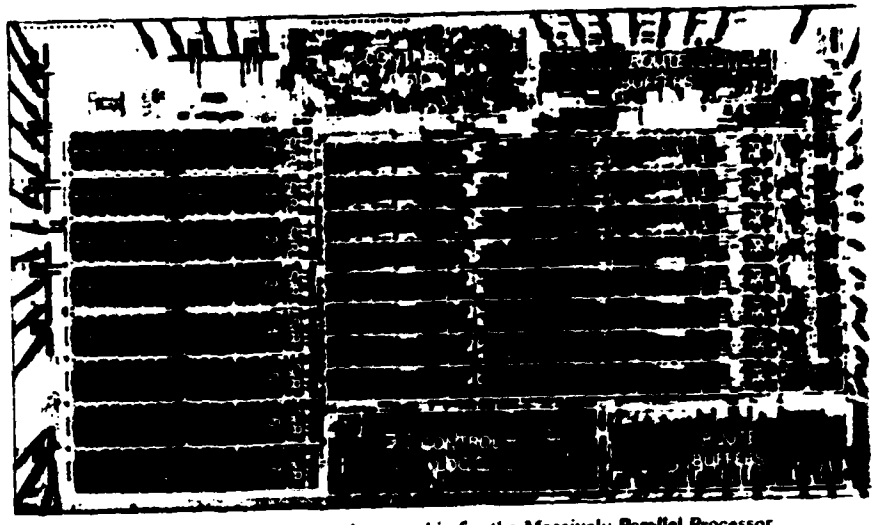


FIG. 2. The processor element chip for the Massively Parallel Processor.

As will be discussed, the planarity problem has several architectural ramifications, so it is important to ask "Why not three-dimensional VLSI?" In fact, three-dimensional VLSI chips (blocks?) are being experimentally developed, and there has been some theoretical work addressing the problem of how to use them and how useful they might be (Preparata, 1983; Rosenberg, 1983). However, even if three-dimensional VLSI research comes to fruition, the planarity problem will not be solved. First of all, most of its ramifications stem from the fact that there are a limited number of dimensions, not from the fact that the limit is two. Without higher dimensionalities, the best we can hope for is to convert the planarity problem into the "solid problem." (This will become apparent in the following discussion). More pragmatically, the use of a two-dimensional

medium allows access via the third dimension—it is possible to replace a chip on a board or a board in a rack; to remove a block from the center of a cube is somewhat more difficult. Thus we accept the plane as a good first approximation for the medium of our architecture, and consider how it influences the design.

4.2.1 The Perimeter Problem

The most critical manifestation of the planarity constraint can be called the "perimeter problem." It refers to the fact that the number of wires leading off a chip is proportional to the perimeter. Because chips are roughly square, the number of external connections is thus proportional to the square root of the area of the chip. Stated another way, if n bits can be transferred onto the chip, through n wires, the number of gates available to process that data is proportional to n^2. If the amount of processing to be performed is at least "quadratic in the amount of data" then the perimeter problem vanishes. But quite often this is not the case: the constraint on the computer designer is to distribute the processing over the chips to make it so (DeRuyck *et al.*, 1982).

The perimeter problem is a direct consequence of our assumption of planarity, but planarity is simply our model for reality, an abstraction that reflects the conditions imposed by a physical situation. To see the actual physical situation that imposes the constraint, we note that chips are not stuck into a computer like so many stamps. Rather, each chip is placed in a package, such as the dual in-line package (DIP), which has an opening large enough to host a chip, and around the inside of this cavity are leads that connect to prongs (or "pins") on two opposing edges of the package. Tiny wires are bonded from the leads to sites on the chip called bonding pads, and a lid is placed over the chip to protect it. Although the bonding pads are traditionally placed on the perimeter of the chip (Fig. 2), this is not the source of the problem. The bonding pads could be placed anywhere on the chip. The perimeter problem arises because the wires are bonded to the package in the same plane as the chip. No matter where the pad is located the wire must cross the perimeter to the lead. The minimum separation of the wires, the number of leads on the inside of the cavity, or the number of pins that can be placed on a package, whichever is smaller, will determine the constant of proportionality for the perimeter problem. Current packages have roughly 100 pins, a number that imposes a severe limitation on chip designers and computer architects.

More sophisticated packaging technology is obviously desirable and is being developed. There are essentially two approaches. One seeks to replace the DIP with variations on the same theme: more leads in the cavity, pins on all sides, pins on the bottom, etc. (Parris and Nelson, 1982). The other, more novel, approach places several chips face down in a carrier so that solder bumps, which replace bonding pads, contact the leads of wires buried in the carrier (Blodgett, 1983). The wires implement a predefined interconnection between the chips and to the pins on the bottom of the carrier in much the same way a printed circuit board does. Both approaches have promise and problems. The first approach allows each chip to be treated as a separate, independent entity, but it is still fundamentally limited by the fact that the wires are in the same plane as the chip; that is, the perimeter problem remains. The second approach relaxes the single-plane limit because the carrier has several layers, but at the cost of a much more specialized structure. Progress with either approach is welcome because pin limitations are very serious, but a satisfactory solution does not appear imminent.

There is one additional aspect of packaging that must be emphasized: improvements in packaging are independent of improvements in device density. We may be offered an improvement in one technology without any improvement in the other technology. This characteristic of the independence of the two technologies is significant chiefly because, historically, packaging has lagged behind density improvements. When this happens the effect of the perimeter problem is compounded: there are more transistors on the chip, but no more pins to deliver additional data. It is then possible to place more processing capacity on a chip than can be used (DeRuyck *et al.*, 1982). It is for this reason that we favor chip designs and architectures that can utilize density improvements without additional pins. Such chips will be said to meet the *strong perimeter constraint;* their external connection requirements are constant with respect to their area.

4.2.2 The Graph-Smashing Problem

The second manifestation of the planarity problem might be called the "graph-smashing" problem. This refers to the usual way of abstracting a circuit as a graph where the vertices represent gates and the edges represent wires. Typically, circuits are not planar graphs in the strict sense; that is, they cannot be drawn in the plane without crossovers. Circuits often exhibit complex interconnection structures, or topologies. When these circuit graphs are projected onto the plane, i.e., "laid out," in VLSI terminology, the "smashed" graph spreads out to occupy a wide area. For example, Fig. 3a shows a crossbar graph abstracting a circuit with four gates (vertices) on the left, each of whose outputs is an input to the four gates on the right; a possible planar layout for this graph is shown in

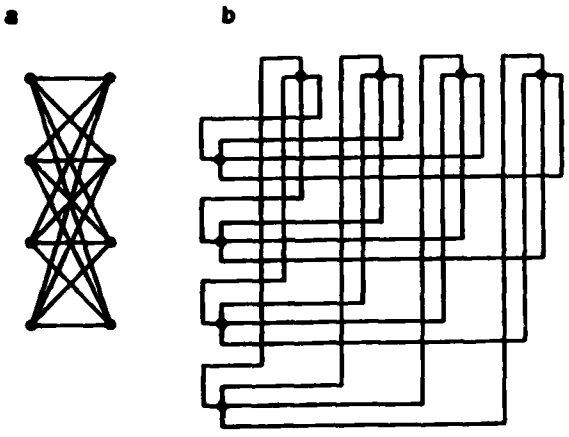


FIG. 3. Crossbar graph (a) and layout (b).

Fig. 3b. The graph must spread out to avoid having several wires crossing over each other at one point (note the four lines intersecting in the middle of Fig. 3a) and to respect the minimum separation required for electrical isolation. The effect is to make wire the dominant component in the layout, thereby reducing the proportion of the chip devoted to active logic. Because there are a multitude of ways of laying out a complex topological structure in the plane, the goal for the VLSI designer is to discover one that is as area efficient as possible.

The graph-smashing problem, though annoying at the chip level, is fairly manageable because the circuits are not large and the data paths tend to be only one wire wide. We have raised the problem in this limited context in preparation for a more in-depth discussion (Section 5) of an analogous situation, the graph-smashing problem at the architectural level. The problem becomes more serious at the architectural level because the vertices of the graph are now the processing elements of a parallel computer and the edges are data paths that can be many wires wide.

5. Architectural Considerations

Having explored some of the architectural ramifications of VLSI at the "low" end of the computer continuum, and having apprised ourselves of some of the advantages and disadvantages of the technology, we now consider the supercomputer end of the continuum. Because supercomputers represent an extreme, an intensive use of components and a maximum achievement of performance, the benefits and limitations of VLSI will come sharply into focus. Our objectives, then, are to consider what architectural options we have for constructing a supercomputer and to assess the impact of VLSI.

The conventional sequential computer, or von Neumann architecture, can be dismissed for reasons that are now widely recognized. Specifically, a von Neumann computer, either with the processor placed entirely on one chip or with the processor sliced up such that it spans many chips, will be subject to the "von Neumann bottleneck." This phenomenon was first recognized by Backus (1978) in connection with programming limitations. It refers to the fact that the instruction stream is inherently sequential; there is only one processing site, and all instructions, operands, and results must flow through a bottleneck between the processor and the memory. There is no distribution of processing activity or control.

Of course, a variety of mechanisms have been invented to reduce the impact of the bottleneck. Caches keep instructions and data "in the processor" so that repeated references do not always require time-consuming memory fetches. Instruction pipelining overlaps the fetch/execute cycle of several instructions at the risk of being thwarted by control jumps. Vector instructions, the main source of today's supercomputers' power, provide a means of packaging a large amount of computation in a form predictable enough for rapid execution. These mechanisms will doubtless be refined and others invented, but they do not remove the bottleneck, they only widen it somewhat.

Future supercomputers must utilize a more decentralized processing strategy, i.e., parallel processing. This suggests using many, perhaps thousands of, processing elements (PEs) as the components of our design. Each PE should perform a complex function; it might be a microcomputer with its own registers and, perhaps, local RAM. The PEs might be placed several to a chip or they might span one or more chips. In any case, because the chips and boards are two-dimensional with limited crossover, we assume as a first approximation that the PEs will be placed in the plane. When we draw in the data paths connecting the PEs together or connecting them to memories, we are confronted with the graph-smashing problem. There are essentially two architectural choices: the common-memory model and the distributed-memory model.

5.1 The Common-Memory Parallel Architectures

The ubiquitous bus structure is an obvious possibility for supporting common memory. Here we imagine a data path (the bus) many wires

wide, stretched out with the PEs and some large memory modules connected to it. As a graph[3] to be smashed into the plane, the bus structure is ideal; it could simply be bent back and forth to form a compact square region in the plane. Moreover, for the case where many PEs reside on one chip, the bus meets the perimeter limitation well because one set of bus wires services the whole chip. From a performance point of view it is considerably less than ideal. The bus enforces exclusive usage, i.e., only one processor can use it at a time, thus it is very likely to be a communications bottleneck when there are many PEs and memories. Perhaps new technologies (e.g., optical fibers) will reduce the bottleneck with very high bandwidth, but, generally speaking, a bus is an unsatisfactory solution when the number of PEs connected to it gets very large.

If usage conflicts doom the bus as a data delivery structure, and if it is intended that each of the PEs has common access to a number of large memory modules, then perhaps each PE/memory pair should have its own connection. This is the crossbar solution illustrated in Fig. 3; it must be dismissed on area occupancy grounds. Specifically, n PEs directly connected to each of m memory modules by data paths of width d will require area proportional to

$$mdn \times ndm = n^2m^2d^2$$

because m groups of wires d wide must emanate from each of n PEs, and n groups must connect to m memories. Taking d as a constant, note that this is a quartic function, a significant constraint on the potential size of the machine.

The crossbar can be simplified somewhat by requiring each PE to connect to only one memory module at a time. (If two PEs want to use the same memory, there is a conflict, and one must wait.) The resulting structure now looks like a grid with switches at the crosspoints, a PE connected to each horizontal data path and a memory connected to each vertical data path. The total area is mnd^2, a quadratic function when d is a constant. This is still large, especially when it is considered that although the switches are very simple, the perimeter problem limits the number that can be packed on a chip when d is very large. (If d is the size of a word, the switches will fit only one per chip with today's packages.) Thus most of the area for the layout will be consumed at the board level where the density is considerably lower.

A common solution for the common-memory parallel computer is to connect the processors to the memories by an *interconnection network*. A typical example is the multistage shuffle-exchange network shown in

[3] Technically, a processor/bus structure is not a graph, but the imprecision will not be harmful here.

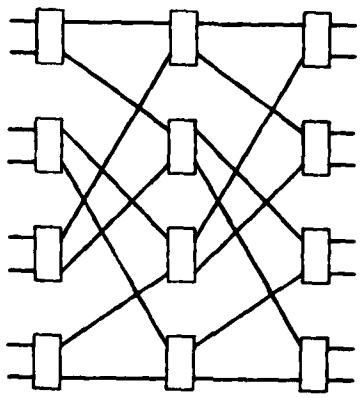


FIG. 4. Shuffle-exchange network.

Fig. 4. The processors are on one side and the memories are on the other side. In the "packet-switched" version, a memory request (fetch or store) from a PE or other switch arrives with the memory module's address attached. Depending on in which stage of the network the switch is located, it interrogates a bit of the address and decides whether to send the request to the upper (0) or lower (1) output data path. Dozens of interconnection networks have been proposed with a variety of properties: alternate topologies, circuit switching rather than packet switching, and differing strategies for handling conflicts. From the VLSI point of view they are essentially the same.

Using the shuffle-exchange network as a representative of the interconnection network solution to common-memory access, let us consider its area occupancy. As a first estimate, we notice that there are $\log_2 n$ stages giving $\frac{1}{2}n \log_2 n$ switches in the network for n PEs. If each switch used a fixed amount of area independent of n, then the switches must occupy area proportional to $n \log_2 n$. (This constant area assumption on switches is not necessarily a good one because there are often buffers in the switches to assist in resolving conflicts, and the buffers must grow modestly in size as n does in order to store the increasingly large addresses.)

The estimate is not good for a more fundamental reason—it ignores the area occupied by the wires. Between each stage, there are $n/2$ wires crossing between the upper half and the lower half of the layout. Because wires must have a constant separation, the distance between each stage is proportional to n. Thus, if we ignore the area occupied by the switches, the area occupied by the wire in this layout is proportional to

$$(n \times n) \times \log_2 n = n^2 \log_2 n$$

or more than the area used by the switches alone. We seek a better solution.

Finding the best layout for the multistage network is complicated, therefore researchers have concentrated on laying out only the shuffle-exchange graph. Such an analysis can give an indication of the area used by each stage. Figure 5 shows that the shuffle-exchange graph is formed by merging two stages of the shuffle-exchange network (a); in (b), the resulting graph is shown with straight exchange edges and curved shuffle edges.

It has been shown (Thompson, 1980) that any layout of the shuffle-exchange graph, using the ground rules we have set forth and ignoring the area of the vertices, must require area proportional to $n^2/(\log_2 n)^2$. This is better than our earlier rough estimate of n^2 for one stage, but for large n it is still a substantial area due to the exponential relationship between $\log n$ and n. Furthermore, it is a lower bound on the area, the best we could do. With some effort, it was shown (Kleitman *et al.*, 1981) that an $n^2/(\log_2 n)^2$ layout can be found for the shuffle-exchange graph.

The conclusion is that the shuffle-exchange graph, when smashed, takes up considerable area, and this is a limitation on its use with VLSI. There are two ways to confront this limitation. First, since so many interconnection networks have been proposed, it could be that others are

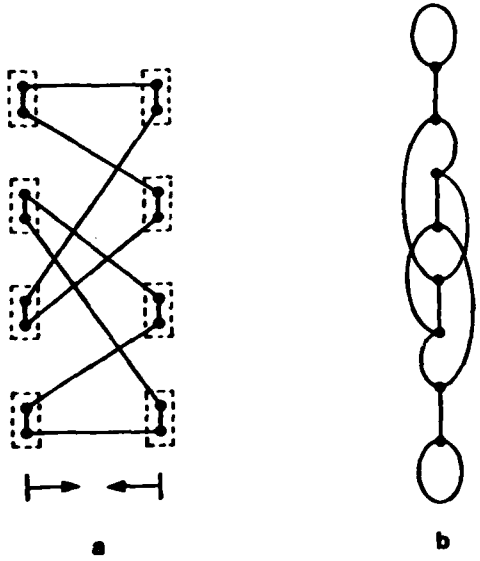


FIG. 5. Shuffle-exchange graph.

substantially more efficient when smashed. This seems to be an unlikely possibility because those alternates that have been analyzed have been found to require comparable area (Preparata, 1983; Wise, 1981). Those that have not been analyzed also may not work because a network with substantial permutation capabilities, a common goal of interconnection networks, is likely to be subject to the same difficulties as was the shuffle exchange. The second way to confront the limitation is to violate the two-dimensional assumption. This has been done for closely related networks to yield three-dimensional organizations (Preparata, 1982; Gottlieb *et al.*, 1983).

5.2 Distributed-Memory Architectures

Another architectural choice for arranging the processors and memories is to give up the common-memory assumption and to couple one processor with one memory module within a PE. The consequence of this approach is that processors have direct access to their own memory, but must communicate with other processors to exchange data. Again the graph is a useful abstraction, where the vertices now represent a processor/memory pair and the edges are communication channels.

The first possibility for placing the PEs in the plane is to use one of the arrangements discussed for the common-memory model. The difference, of course, is that previously each vertex of the graph represented a processor or a memory; now each vertex will represent a processor and a memory. The distinction does not much change our conclusions about those arrangements: the layout of the bus is still area efficient, but the congestion problem remains; the crossbar is just as inefficient of area; the shuffle-exchange graph, rather than the whole network, will suffice for our coupled processor/memory situation, but its area requirements are still large. Our conclusions about area usage did not change because our area analysis was based on the wire area.

Because it is unnecessary to support the common-memory access, we can consider other graphs that have efficient planar layouts. The two most frequently discussed choices are the array and the binary tree. We consider each in turn.

5.2.1 *The Array Architectures*

The array structure is an arrangement where the PEs occupy points in the plane with integer coordinates, and the connections are to nearest neighbors. There are four-, six-, and eight-connected versions of the array in which the PEs are connected to their four, six, or eight nearest neigh-

bors. (The six-connected case consistently adds either the northwest–southeast or the northeast–southwest connections to the four-connected case.) There is also a two-connected instance, called a linear array, with somewhat different characteristics: when the two end PEs are the only PEs connected to external I/O, the structure is limited by the amount of data it can receive. When all the PEs are connected to external I/O, it cannot be smashed into the plane very well because the number of wires is proportional to n and by the perimeter constraint it must use area proportional to n^2 when laid out in a square; in either case the data transmission along the array is very slow. We will not consider the linear array further.

Taking the area of the PE as fixed, the area occupied by n PEs arranged in an array is proportional to n. This is the best possible area utilization and it also has the property of efficient PE-to-PE communication. However, there is also a limitation: data can enter and leave the array only from the perimeter PEs; PEs in the interior must receive their data from their neighbors by "bucket brigade." Note that this is not an instance of the perimeter problem. The perimeter constraint would predict that an n PE array could only have external data paths proportional to $\sqrt{n}$ in number. There are $4\sqrt{n}$ data paths for the four-connected array, so the perimeter constraint is met. The fact that only perimeter PEs are connected to the external data source is a programming limitation. It complicates using the architecture.

There is a class of ingenious architectures that accommodate the bucket brigade paradigm of data motion very well. They are the systolic arrays (Kung, 1982). Data enters the systolic array at the perimeter. Each PE is very simple, with a few registers and a simple processor capable of performing an arithmetic expression. The systolic PEs generally execute a single instruction stream: they read values from two or three directions, evaluate a scalar arithmetic expression, and write out their input values and results to their neighbors. Usually, a systolic array can complete the computational tasks involving a particular data value in the time it takes for it to move across the array. These properties—the ability to utilize data input only at the perimeter, the ability to utilize only nearest-neighbor connections, the ability to employ only a single instruction stream, and the fact that problems are solved in time proportional to the time it takes to read the input—make the systolic array algorithms extremely effective in terms of exploiting VLSI.

5.2.2 *Tree Architectures*

The binary tree structure is a PE interconnection arrangement that is convenient for evaluating the kind of algorithms Backus proposed as an alternative to the von Neumann style. The conventional way of drawing an

$$n = 2^k - 1$$

node complete binary tree uses area proportional to $n \log_2 n$, because half of the nodes are leaves and its height, k, is $\log_2 n$. However, there is a more area-efficient embedding of a complete binary tree that requires area proportional to n. The embedding, shown in Fig. 6, is formed inductively as follows. Suppose we have two complete binary trees of height h, each laid out in a region of the plane. (Of course, a single node suffices for the height-1-basis case.) Suppose further that each region also contains a spare node. Then we can join the two trees as subtrees of one of the spare nodes such that it becomes the root of a complete binary tree of height $h + 1$. The other spare node and our new, higher, complete binary tree fulfill the induction hypothesis. That the result can be made to occupy only linear area is clear from Fig. 6 even though the exact construction algorithm is somewhat involved (Snyder, 1982).

In addition to the efficient area usage, this inductive definition has another advantage: the complete binary tree satisfies the strong perimeter constraint. Recall that as device density improves, the perimeter constrains the number of wires coming onto the chip, thus in the absence of improved packaging there can be no improvement in the number of available external connections. The binary tree requires only a constant number (four) of external wires, no matter how large the subtree is on the chip. To see this, suppose a tree of height h and its spare fit on a chip. When we join two chips to form a height $h + 1$ tree, the chip containing the root will use three connections: one to the on-chip subtree (thus no external connection is used), one to the other subtree, and one to the parent, if any. The chip with the spare uses only one wire to the other chip. When this pair is combined with a like pair to form a height $h + 2$ tree, the chip with the spare gets three more connections: one each to its

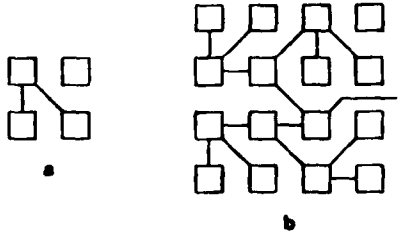


FIG. 6. Binary tree construction-basis step, $h = 2$ (a), and induction step, $h = 4$ (b).

two subtrees, because there is not an on-chip subtree at this or any subsequent step, and one to the parent. Because this discussion is independent of any particular value for h, we see that a density improvement that allows a much larger tree on each chip will not require any more external connections.

There are some problems with this binary tree layout, however. In general, the perimeter cannot always be connected to the leaves. This makes it a difficult architecture to program because many "tree-based" algorithms presume that data are either read or written at the leaves. We must apparently give up the efficient area utilization to have leaf connections: any convex layout that provides external connections to the leaves of a complete binary tree must have area proportional to at least $n \log_2 n$ (Brent and Kung, 1980). A second problem is that the layout contains wires whose length is proportional to $\sqrt{n}$. Long wires can cause propagation delay. There is an alternate layout that achieves the theoretical minimum of $\sqrt{n}/\log_2 n$ for a linear area embedding (Paterson *et al.*, 1981). If the leaves are to be externally connected, the best possible length is $n/\log_2 \log_2 n$, and the area must be proportional to $n \log_2 n$. Finally, there is a bottleneck at the root of the tree that limits the amount of data that can pass between the two halves.

Obviously, there are multitudes of other graph structures that could be used to connect the PEs together, each with its own architectural features. There is no need to consider any others in detail, for the constraints of planarity should now be clear. Smashing a graph can have a dramatic effect on the area usage, as well as secondary effects such as increased wire length.

6. The CHiP Architecture

Having enumerated some of the advantages of VLSI that can be exploited and some of the disadvantages that should be avoided, we come to the matter of striking a balance between the two to build an effective supercomputer. In this section we use the Configurable, Highly Parallel (CHiP) computer (Snyder, 1982) as a case study for how the balance might be struck. The CHiP architecture has been designed with many of the foregoing considerations in mind, and although it does not utilize all of the benefits nor avoid all of the limitations, it represents one complete design in which the trade-offs have been evaluated and decided upon. Before analyzing how effectively the CHiP machine balances the trade-offs, we first describe it.

6.1 CHiP Components

The most important component of a CHiP computer is its *lattice*. The lattice is a two-dimensional structure of programmable switches connected by data paths into which processing elements have been placed at regular intervals. Figure 7 shows two examples; squares represent PEs, circles represent switches, and lines represent data paths. A production CHiP machine may have thousands of processors.

The PEs are microprocessors, each coupled with several thousand bytes of RAM for program and data storage. PEs would likely have floating-point capability because most problems worthy of highly parallel computation are numerical in nature. Each PE has its own instruction counter. There is a common clock, and PEs with instructions to execute begin executing them simultaneously. (The exact details of the synchronization, however, are too complex to justify treatment here.) Data can be read or written through any of the eight data paths or *ports* connected to the PE. The unit of data transmission is generally a word, though the physical data path may be narrower. The PEs operate in a "data-driven" mode: reads wait until the arrival of data; writes take place immediately unless they would cause "buffer overrun," in which case they idle until space is available. Note that PEs are not directly connected to each other; they are connected only to switches.

The switches are programmable devices, each with a small (~16 words) local RAM. The switch memory is used to store instructions (one per word) called *configuration settings*. Each configuration setting specifies

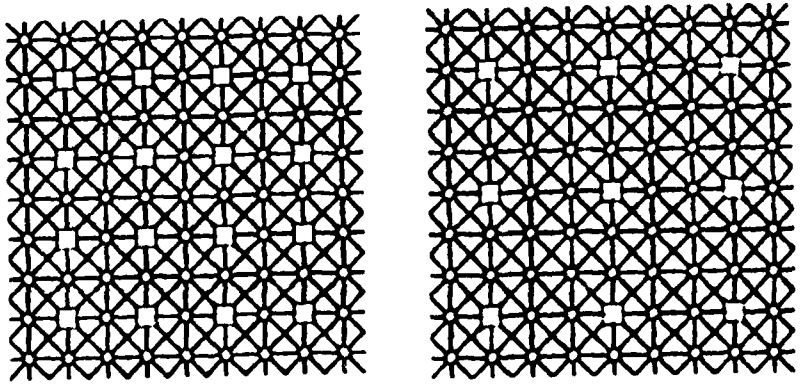


FIG. 7. CHiP lattices.

pairs of data paths to be connected. Each pair, called a *crossover level*, forms a direct, static data path across the switch that is independent of the others. The data path is bidirectional, full duplex, i.e., data can move in either direction simultaneously. Executing a configuration setting causes the specified connections to be established and to persist over time, e.g., over the execution of an entire algorithm.

To connect the PEs together, the programmer *configures* the lattice. That is, he programs each switch such that, collectively, they implement the desired processor interconnection graph. Figure 8 illustrates three examples of how the lattice of Fig. 7a might be configured to implement some commonly used interconnections. Note that the binary tree and mesh connection use only a single crossover level, but the torus uses two. Drawn in the usual way, a torus is a mesh with its left- and right-edge PEs connected and its top- and bottom-edge PEs connected. In Fig. 8 the torus has been "interleaved" such that the maximum data path length is reduced from being proportional to $\sqrt{n}$ in the naive configuration to at most 3.

In addition to the lattice, the CHiP architecture has a controller to manage the computation. The controller is a conventional sequential computer that is used for loading programs into the PE memories and configuration settings into the switch memories. (This loading is done using an additional interconnection called the *skeleton;* this is not shown in Fig. 8 and will not be mentioned further.) The controller delivers the external data to switches on the perimeter of the lattice. The controller performs other operations, but to understand them we must consider the nature of parallel solutions to large problems.

6.2 An Example of CHiP Computation

Large problems worthy of parallel computation are rarely solved by an algorithm with a single communication structure. Rather, the problem tends to divide into parts, each of which has a characteristic data motion. One of these units is called a *phase*, an algorithm with a single interconnection structure. The goal in CHiP programming is to program the separate phases and then to compose them into a problem solution. The controller assists in this composition activity. Consider an example.

Suppose we wish to find the solution to an elliptic partial differential equation. The task is divided into three phases: an input phase, an approximation phase using, for example, the successive overrelaxation (SOR) method, and a termination phase. The input phase reads the initial values from external memory into each PE. To speed up this process, we read data for each row of PEs from two channels on the left and two

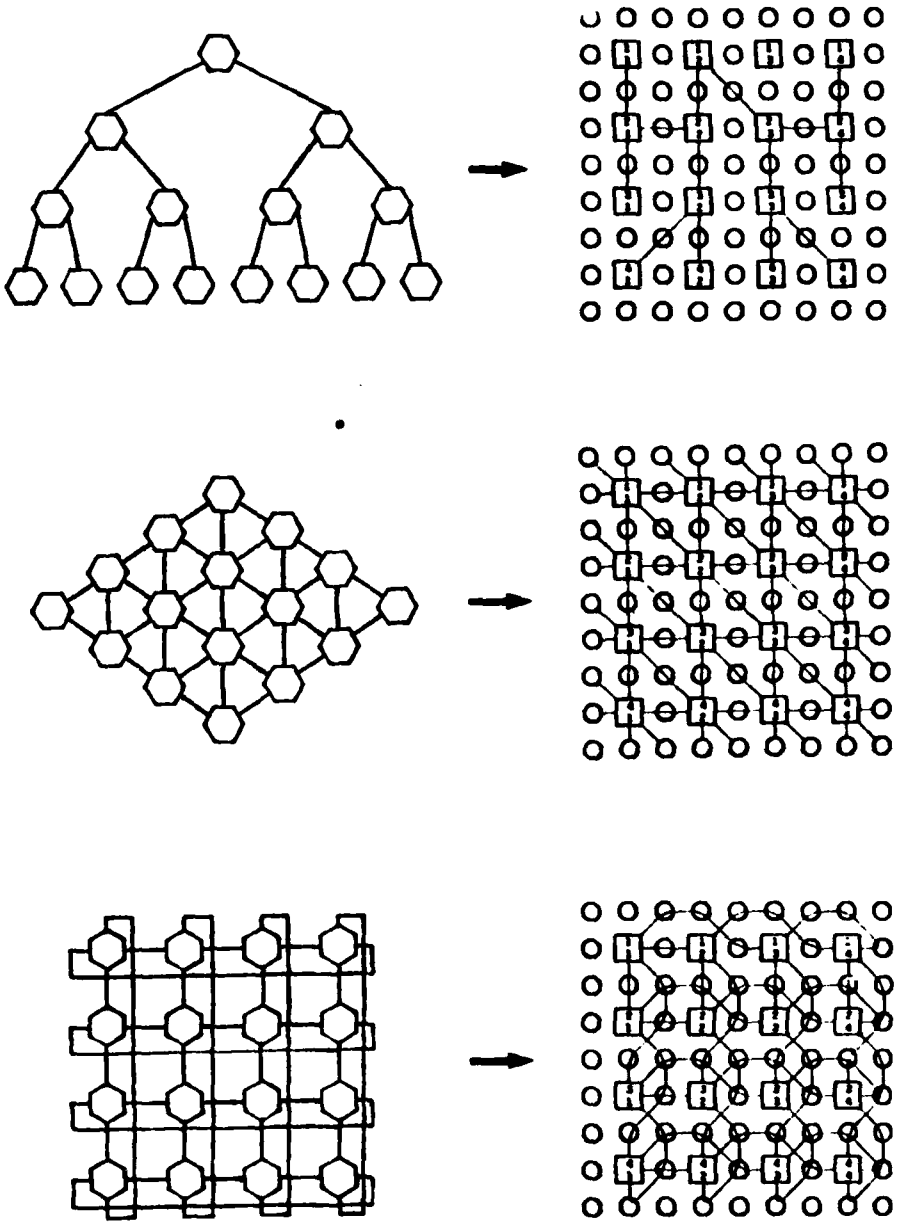


FIG. 8. Embedding graphs into the lattice of Fig. 7a.

channels on the right, as shown in the interconnection structure of Fig. 9a. The code segment for each PE reads data in a bucket brigade fashion. Note that some care must be taken to ensure the data are sent to the lattice in the right order.

The second phase uses the SOR method to compute a numeric approximation, the details of which are unimportant for our discussion. Suffice it to say that at each step the PEs read the current approximation from their four nearest neighbors and compute a new approximation based on those values. They also compute an error term. Clearly, the proper interconnection structure for this phase is illustrated in Fig. 9b.

The third phase involves determining whether the maximum error term over all of the PEs is sufficiently small. Because finding the maximum of a set is a commutative and associative operation, the binary tree is a suitable interconnection structure. (Figure 9c shows a binary tree which has a root with only a single child; this solves the problem that there are 64 PEs, but a height 6 complete binary tree has only 63 nodes.) The PE code segments perform as follows: leaves pass their error terms to their parents, and internal nodes pass to their parents the maximum of their and their children's error terms. The controller receives the maximum from the root.

The computation proceeds as follows. The controller down-loads into each PE the three appropriate code segments for each PE. Not every PE receives the same three because they play different roles during each phase. An edge PE might get a generic input code, a "boundary" SOR code, and a leaf code, while an interior PE might get a generic input code, an "interior" SOR code, and a leaf code, etc. At the same time that the PEs are being loaded, the controller loads the configurations into the switch memories. The proper configuration setting for the phase 1 interconnection structure might be loaded into location one of each switch, the four-connected mesh configuration setting into location two, and the tree settings into location three.

When loading is complete, the controller broadcasts to all switches a command to execute the configuration setting in location one. This causes the lattice to be configured into the structure of Fig. 9a. The first code segment is then invoked and the PEs begin reading data. The input phase completes when the data are read. The controller then broadcasts to the switches to execute the configuration setting in location two. In one logical step the communication structure is rearranged into the four-connected mesh. The second code segment is invoked and the PEs begin the SOR algorithm. The controller allows them to execute a fixed number of iterations and then broadcasts for phase 3 to begin. After roughly $\log_2 n$ steps, the controller will receive the maximum error term. If this is suffi-

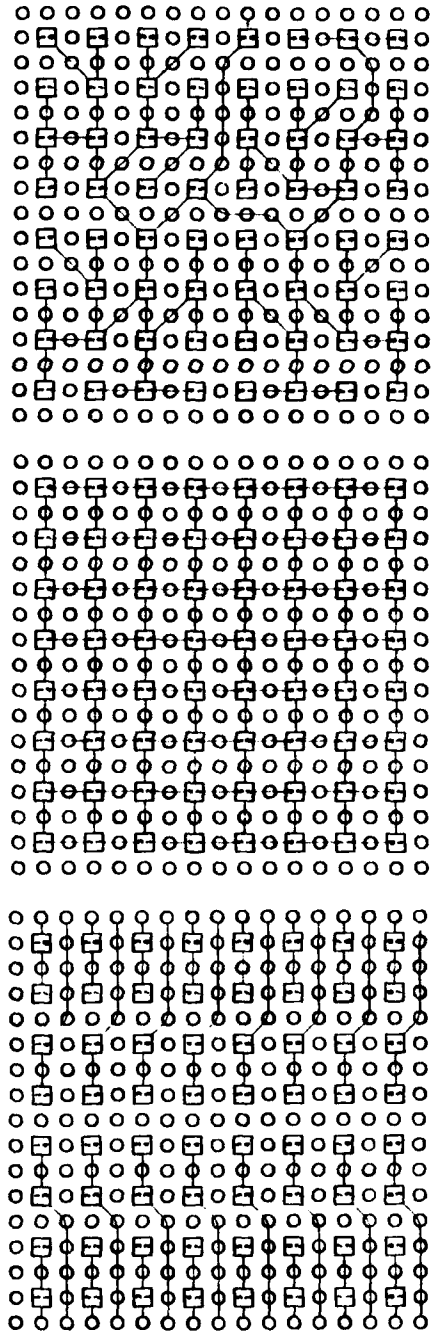


FIG. 9. Interconnection patterns for an elliptic p.d.e. algorithm.

ciently small the computation is complete. If it is not small enough, phase 2 and phase 3 are repeated. (Clearly, the phase 1 configuration can be used with an appropriate code segment to output the results.)

6.3 Characteristics of the CHiP Machine

Some comments about the preceding example are in order. First, the interconnection structure was imposed on the lattice externally. The PEs were not responsible for determining the source/target of their input/output. They simply transferred data via a port, and the interconnection structure provided the route. Second, because all of the communication paths are point-to-point, there is no data transmission bottleneck due to two paths sharing a segment. There was a bottleneck in getting the data into the lattice because the data enter at the perimeter, and all PEs require data. Third, the elliptic p.d.e. algorithm would change very little with a larger lattice. The graphs would change, but all generalize naturally. The *types* of PE codes—input code, "boundary" SOR code, leaf code, etc.—would not change; there would simply be more instances. The controller code would not change in concept.

There are certain architectural features of the CHiP machine that require further explanation. The size of the lattice, given by the number of processors in each dimension, could be any number $x \cdot y$. Experience indicates that many embeddings are simplified if

$$x = y = 2^k$$

Furthermore, many algorithms seem to benefit (and few are harmed) by having the greater processor capacity provided by a square rather than by the greater perimeter (and thus greater I/O) that a rectangle would permit. A possible exception might be database applications which could utilize an $n/\log_2 n \times \log_2 n$ aspect ratio and might benefit from the greater perimeter.

The exact value of the corridor width, the number of switches separating two adjacent PEs, has not been specified because this feature distinguishes different CHiP family members. Choosing a particular width has some interesting consequences. Clearly, greater corridor width implies greater area devoted to switches and less to processing capacity. Because the power of the computer derives from the number of processors, we are inclined to reduce corridor width to raise processor density. We will not reduce the width to zero, i.e., hardwire the PEs together, however. Switches provide flexibility by relieving the programmer from having to circumvent a hardwired communications structure each time his algorithm fails to match the given structure. As we saw in the example of the solution to an elliptic p.d.e., this can be frequent. Furthermore, the

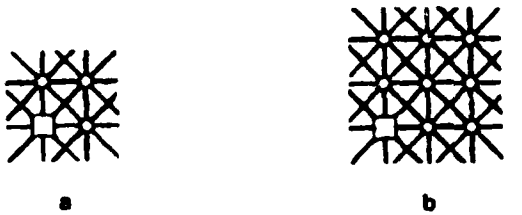


FIG. 10. Corridor width increase.

switches do not use very much area. Their exact area requirements are influenced most by the data path width, but for small data path widths they might use only 1/100 the area of a PE. A more serious concern with switches is that they increase the "pin count," the number of wires leading on and off the chip. To see this, suppose one PE, the corridor above it, and the one on its right side fit on one chip (see Fig. 10). As the width increases, the number of wires leading from the region increases by 12 times the width of the data path for each unit increase in corridor width. This is consistent with the perimeter limitation, but it is still significant. Another worry is that with each added switch there is an added propagation delay of passing a signal through a few transistors. The delay per switch is quite small, but for large corridors it cannot be ignored.

On the side of larger corridors, it can be argued that they provide greater flexibility for embedding complex graphs. This is true from experience, and it can be proved theoretically. Recall that the shuffle-exchange graph, which might be used for computing Fourier transforms, requires area proportional to $n^2/(\log_2 n)^2$. Translated into lattice terms, a corridor width proportional to at least $\sqrt{n}/\log_2 n$ would be required to embed the shuffle exchange *and* use all the PEs. Anything less would leave PEs unused. Another advantage is that wide corridors permit the use of several interconnection structures simultaneously. For example, we might have overlaid the binary tree with our mesh in the example. Having multiple configurations in force simultaneously reduces the number that must be stored, thus reducing the memory (and therefore the area) needed for each switch, thereby helping to pay for the added width. But to make effective use of this strategy would likely require more ports.

How all of these considerations balance out is unclear. Early experience suggests that a width in the range from one to four is realistic.

7. Evaluating the CHiP for VLSI

With the advantages and disadvantages of VLSI in mind we analyze how the CHiP architecture stacks up.

Clearly, the architecture is regular. Visualize building a CHiP computer out of chips containing a region of the lattice plus the switches of the corridors on the top and to one side of the region (see Fig. 10.) These chips can be placed side by side to construct the entire lattice except for one side and the bottom perimeter corridors. A separate chip type would complete the design. These chips also achieve a high level of integration because the active logic, memory, and switching capability are uniformly distributed throughout the lattice.

In terms of device density, there is a mixed review. Certainly, the architecture uses relatively little wire because of the nearest-neighbor design of the switches and PE connections, therefore the technologically available device density is reasonably well used. As for architectural strategies for increasing effective density, we can point to the switches. They are simple, circuit-switched devices that implement an externally imposed, static interconnection structure. This factors out much of the routing complexity. On the other hand, each PE contains a copy of a program, and to the extent that the algorithms use the same codes for each PE, this is redundant. For example, a systolic array algorithm running on the CHiP machine would likely have each PE contain the identical code. Thus one form of architectural density increase has been forgone for flexibility.

The lattice meets the perimeter constraint satisfactorily. Every square region of size n requires connections proportional to $\sqrt{n}$. Furthermore, for narrow corridors and a narrow data path, the constant of proportionality is reasonably small. The planarity problem has also been addressed. The lattice uses linear area and the switches provide a maximum of four levels of crossover. The maximum data path length for any configuration corresponds to whatever it would be in a planar embedding. In particular, local connections are short, and extremely nonlocal connections have a propagation time proportional to their length.

We have already observed that the CHiP computer uses available device density fairly effectively. But how good is it at exploiting the improvements in device density due to reduced feature size? We have observed that the CHiP computer meets the perimeter constraint, but to exploit improved density without improved packaging requires it to meet the strong perimeter constraint. Whether or not the CHiP PEs meet this constraint depends on how many PEs fit on a chip. For a specialized CHiP machine with tiny PEs that pack several to a chip, the architecture does not meet the strong perimeter constraint. Any improvement in device density would permit a larger portion of the lattice to fit on a chip and thus would require more external connections that may not be available without better packaging.

For general-purpose CHiP computers matters are not as serious. To see this we observe that with today's technology it would take considerable ingenuity to place on a chip even one PE of the type specified in Section 6. The day when this will happen, however, is probably not far off, so postulate that it can be done. Then consider a 50% reduction in feature size. From the preceding paragraph it is clear that placing a four-PE lattice region on the chip is not the proper strategy. However, using the additional area for more memory is quite reasonable because it leaves the number of external connections unchanged, i.e., it meets the strong perimeter constraint. (Early experience with programming the CHiP computer indicates that additional on-chip memory will be an effective use of density improvements for the foreseeable future.) Thus, unlike the situation discussed in Section 2, where added on-chip memory was not very helpful for a multichip von Neumann architecture, the PEs of a CHiP computer can use the memory.

The conclusion seems to be that for the advantages and disadvantages of VLSI discussed here, the CHiP architecture is reasonably responsive.

8. Summary

Our discussion has touched on a variety of topics: the economics of mass production, the geometry of the plane, the architecture of supercomputers. It is time to distill the main points.

First there was the single-chip microprocessor discussion, our first chance to consider the consequences of using VLSI for computer architecture. The outcome of that analysis was that the benefits of VLSI, especially low cost, do not suggest a general trend toward single-chip computers. This somewhat surprising outcome motivated a more careful analysis of both the advantages and disadvantages of the technology. The major advantages listed were low cost, a high degree of integration, and the potential for still greater device density. The chief disadvantages were referred to as the perimeter problem, the strong perimeter constraint, and, most importantly, the planarity problem. With these characteristics of VLSI in mind, we considered how both common-memory and distributed-memory parallel computers might be influenced by the use of VLSI. The main cost, that of interconnecting the system components, was seen to vary widely with the differing architectural choices. Finally, the CHiP computer was introduced as a specific instance of a VLSI-compatible architecture; it was then examined in terms of its exploitation of the benefits of VLSI and its avoidance of the liabilities.

In summary, to exploit the benefits of VLSI technology and simulta-

neously avoid its limitations require us to replace the familiar, tested architectural structures with new, imaginative architectures.

Acknowledgments

It is a pleasure to thank Francine Berman, Janice Cuny, Dennis Gannon, and Steven Holmes for their constructive comments on early drafts of this manuscript, and Julie Hanover for her patient preparation of the manuscripts. I would also like to express my appreciation to members of the Blue CHiP Project who have been a delight to have as colleagues. Support for the research on the CHiP Computer has been provided by the Office of Naval Research under Contracts N00014-80-K-0816 and N00014-81-K-0360, the latter is SRO-100.

References

Backus, J. W. (1978). Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Commun. ACM* 21(8), 613–641.

Batcher, K. E. (1980). Design of a massively parallel processor. *IEEE Trans. Comput.* C-29(9), 48–56.

Blodgett, A. J., Jr. (1983). Microelectronic packaging. *Sci. Am.* 249(1), 86–96.

Bouknight, W. J., Denenberg, S. A., McIntyre, D. E., Randall, J. M., Sameh, A. H., and Slotnick, D. L. (1972). The Illiac IV system. *Proc. IEEE* 60(4), 369–379.

Brent, R. P., and Kung, H. T. (1980). On the area of binary tree layouts. *Inf. Process. Lett.* 11(1), 46–48.

Clark, W. A. (1980). From electron mobility to logical structure: A view of integrated circuits. *ACM Comput. Surv.* 12(3), 325–356.

DeRuyck, D. M., Snyder, L., and Unruh, J. D. (1982). Processor displacement: An area-time trade-off method for VLSI. *In* "Proceedings of the MIT Conference on Advanced Research in VLSI" (P. Penfield, ed.), pp. 182–187. Artech House, Dedham, Maine.

Fitzpatrick, D. T., Foderaro, J. K., Katevenis, M. G. H., Landman, H. A., Patterson, D. A., Peek, J. B., Peshkess, Z., Séquin, C. H., Sherburne, R. W., and Van Dyke, K. S. (1981). A RISCy approach to VLSI. *VLSI Des.* 2(4), 14–20.

Foderaro, J. K., Van Dyke, K. S., and Patterson, D. A. (1982). Running RISCs. *VLSI Des.* 3(4), 27–32.

Gottlieb, A., Grishman, R., Kruskal, C. P., McAuliffe, K. P., Rudolph, L., and Snir, M. (1983). The NYU Ultra computer-designing an MIMD shared memory parallel computer. *IEEE Trans. Comput.* C-32(2), 175–189.

Hedlund, K. S., and Snyder, L. (1982). Wafer scale integration of configurable, highly parallel processors. *Proc. Int. Conf. Parallel Process., IEEE* pp. 262–264.

Kleitman, D., Leighton, F. T., Lepley, M., and Miller, G. L. (1981). New layouts for the shuffle-exchange graph. *Proc. 13th Annu. Symp. Theory Comput., Assoc. Comput. Mach.* pp. 278–292.

Kung, H. T. (1982). Why systolic architectures? *IEEE Comput.* 15(1), 37–46.

Lincoln, N. R. (1983). Supercomputers = colossal computations + enormous expectations + renowned risk. *IEEE Comput.* 16(5), 38–47.

Mead, C., and Conway, L. (1980). "Introduction to VLSI Systems." Addison-Wesley, Reading, Massachusetts.

Moore, G. E. (1979). Are we really ready for VLSI? *Proc. Caltech Conf. Very Large Scale Integration, 1979* pp. 3–14.

Parris, S. R., and Nelson, J. A. (1982). "Practical considerations in VLSI packaging. *VLSI Des.* 3(4), 44–49.

Paterson, M. S., Ruzzo, W. L., and Snyder, L. (1981). Bounds on minimax edge length for complete binary trees. *Proc. 13th Annu. Symp. Theory Comput., Assoc. Comput. Mach.* pp. 293–299.

Patterson, D. A., and Séquin, C. H. (1981). RISC I: A reduced instruction set VLSI computer. *Proc. Int. Symp. Comput. Archit., 8th, 1981.*

Preparata, F. P. (1982). "Algorithm Design and VLSI Architecture," Tech. Rep. Coordinated Sciences Laboratory, University of Illinois, Urbana.

Preparata, F. P. (1983). Optimal three dimensional VLSI layouts. *Math. Syst. Theory* 16, 1–8.

Raffel, J. E., Naiman, M. L., Burke, R. L., Chapman, G. H., and Gottschalk, P. G. (1980). Laser programmed vias for restructurable VLSI. *Proc. Int. Electron Devices Meet., IEEE* pp. 132–135.

Rosenberg, A. L. (1983). Three-dimensional VLSI I: A case study. *J. Assoc. Comput. Mach.* 30(3), 397–416.

Schwartz, J. T. (1980). Ultracomputers. *ACM Trans. Program. Lang.* 2(4), 484–521.

Siskind, J. M., Southard, J. R., and Couch, K. W. (1982). Generating custom high performance VLSI designs from succinct algorithmic descriptions. *In* "Proceedings of the MIT Conference on Advanced Research in VLSI" (P. Penfield, ed.), pp. 28–40. Artech House, Dedham, Maine.

Snyder, L. (1982). Introduction to the configurable highly parallel computer. *IEEE Comput.* 15(1), 47–56.

Swensson, C. (1983). VLSI physics. *Integration* 1(1), 3–19.

Thompson, C. D. (1980). A complexity theory for VLSI. Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Werner, J. (1981). Software for gate-array design: Who is really aiding whom? *VLSI Des.* 2(4), 22–32.

Wise, D. S. (1981). Compact layout of Banyan/FFT networks. *In* "Proceedings of the Carnegie-Mellon Conference on VLSI Systems and Computations" (H. T. Kung, R. Sproull, and G. Steele, eds.), pp. 186–195. Computer Science Press, Washington, D.C.